Numerical Methods for Partial Differential Equations
A.Y. 2025/2026

Laboratory 01

The finite element method for the 1D Poisson equation

Exercise 1.

Let $\Omega = (0, 1)$. Let us consider the Poisson problem

$$\begin{cases} -\left(\mu(x)\, u'(x)\right)' = f(x) & x \in \Omega = (0,1) \qquad \text{(1a)} \\ u(0) = u(1) = 0 & \qquad \text{(1b)} \end{cases}$$

with $\mu(x) = 1$ for $x \in \Omega$, and

$$f(x) = \begin{cases} 0 & \text{if } x \leq \dfrac{1}{8} \text{ or } x > \dfrac{1}{4}\,, \\ -1 & \text{if } \dfrac{1}{8} < x \leq \dfrac{1}{4}\,. \end{cases}$$

1.1. Write the weak formulation of problem (1).

1.2. Write the Galerkin formulation of problem (1).

1.3. Write the finite element formulation of problem (1), using piecewise polynomials of degree r, and write the associated linear system.

1.4. Implement in `deal.II` a finite element solver for (1), using piecewise polynomials of degree $r = 1$ and with a number of mesh elements $N_{\mathrm{el}} = 20$.

Exercise 2.

Consider again the general Poisson problem (1), with $\mu = 1$ and $f(x) \in L^2(\Omega)$ a generic function.

2.1. Assuming that the exact solution to (1) is $u_{\text{ex}}(x) = \sin(2\pi x)$, determine the expression of $f(x)$.

2.2. Starting from the solution of Exercise 1, implement a new method that computes the $L^2(\Omega)$ or $H^1(\Omega)$ norm (depending on the input argument) of the error between the computed solution and the exact solution:

$$e_{L^2} = \|u_h - u_{\text{ex}}\|_{L^2} = \sqrt{\int_0^1 |u_h - u_{\text{ex}}|^2 \, dx} \,,$$

$$e_{H^1} = \|u_h - u_{\text{ex}}\|_{H^1} = \sqrt{\int_0^1 |u_h - u_{\text{ex}}|^2 \, dx + \int_0^1 |\boldsymbol{\nabla} u_h - \boldsymbol{\nabla} u_{\text{ex}}|^2 \, dx} \,.$$

The new method should have the signature

```
double
Poisson1D::compute_error(
  const VectorTools::NormType &norm_type,
  const Function<dim>         &exact_solution) const
```

2.3. With polynomial degree $r = 1$, solve the problem (1) with finite elements, setting $N_{\text{el}} = 10, 20, 40, 80, 160$. Compute the error in both the $L^2(\Omega)$ and $H^1(\Omega)$ norms as a function of the mesh size h, and compare the results with the theory.

2.4. Repeat the previous point setting $r = 2$.

2.5. Let us now redefine the forcing term as

$$f(x) = \begin{cases} 0 & \text{if } x \leq \dfrac{1}{2} \,, \\ -\sqrt{x - \dfrac{1}{2}} & \text{if } x > \dfrac{1}{2} \,. \end{cases}$$

The exact solution in this case is

$$u_{\text{ex}}(x) = \begin{cases} Ax & \text{if } x \leq \dfrac{1}{2} \,, \\ Ax + \dfrac{4}{15}\left(x - \dfrac{1}{2}\right)^{\frac{5}{2}} & \text{if } x > \dfrac{1}{2} \,, \end{cases}$$

$$A = -\frac{4}{15}\left(\frac{1}{2}\right)^{\frac{5}{2}} \,.$$

Check the convergence order of the finite element method in this case, with polynomial degrees $r = 1$ and $r = 2$. What can you observe?

Possibilities for extension

Reading parameters from file. In the simple implementation proposed here, all the problem parameters (the coefficient μ, the number of mesh elements, ...) are part of the source code. Therefore, to change them, a user would need to modify and recompile the program. To avoid this, `deal.II` offers a convenient interface for reading parameters from an external text file, without the need to recompile, through the class `ParameterHandler`. With the help of `deal.II`'s documentation and tutorials, add support for reading parameters from file to the program of Exercise 1.

Profiling. After writing and testing a program, we usually want to measure its performance to find bottlenecks and possibly introduce optimizations. `deal.II` offers the class `TimerOutput` for a simple way of profiling the code. With the help of `deal.II`'s documentation, add profiling to the program of Exercise 1, to measure the execution time of initialization, system assembly and system solution. (For more sophisticated profiling, you can check out `gperftools` at `https://github.com/gperftools/gperftools`).

Patch test. Repeat Exercise 2 choosing an exact solution that can be represented by the finite element space, and observe the behavior of the error as the mesh is refined.

Parsing functions from file. As seen above, large simulation programs usually allow problem parameters to be defined in an external file (as opposed to hard-coded into the source files). This can also be done for functional data. `deal.II` implements a class named `FunctionParser` for this precise purpose. With the help of `deal.II`'s documentation, modify the code from Exercise 2 so that it reads all relevant parameters from file, including the definitions of functional data.

References